Project report: Squashpongthingy

Daniel Raneland (dt05dr1@student.lth.se)
Farooq Anwar (sx08fa9@student.lth.se)
Christopher Abrahamsson (dt05ca3@student.lth.se)

2009-10-19

1 Abstract

The aim of this project was to create a game based on ideas from pong brought into the third dimension.

The game takes place in a room with 3 walls, a floor and a roof. The player controls a racket positioned where the fourth wall would be.

The aim of the game is to keep a ball bouncing around in the room from escaping the room through the missing fourth wall, every time the player prevents the ball from escaping a point is awarded. The game ends when the ball escapes the room.

Contents

2 Architecture

To create this game we needed a game engine, a mouse to control the racket and a screen to display the game on. This was implemented by using a customized VGA Controller, a pre-made PS/2 Controller and a game engine written in software running on a Microblaze softcore processor. An overview of the architecture can be seen in figure 1.

3 Hardware

The hardware used in the project is a customized VGA Controller and a reused PS/2 Controller from Xilinx. The block diagram can be seen in figure 2.

3.1 VGA Controller

The VGA Controller is a modified VGA Controller we found on Open Cores[1]. It functions by iterating over every pixel displayed and using various methods defined to determine the colour of the pixel. There is one method call for each primitive seen on the screen and by having different priorities for the methods we can be sure that the ball and racket will always be visible and that graphical artifacts due to the background overrwiting the foreground will not exist.

To enable communications between the VGA Controller and the game engine we created an IP-Core from the VGA Controller in the EDK by using a wizard, the wizard created a PLB[2] interface for us and all we had to do was to connect the VGA Controller with the PLB interface and make sure that the VGA Controller picked up new values from the bus when they arrived from the software.

3.1.1 Graphics component

The graphics component consists of a rectangle function, a circle function aswell as a digit process.

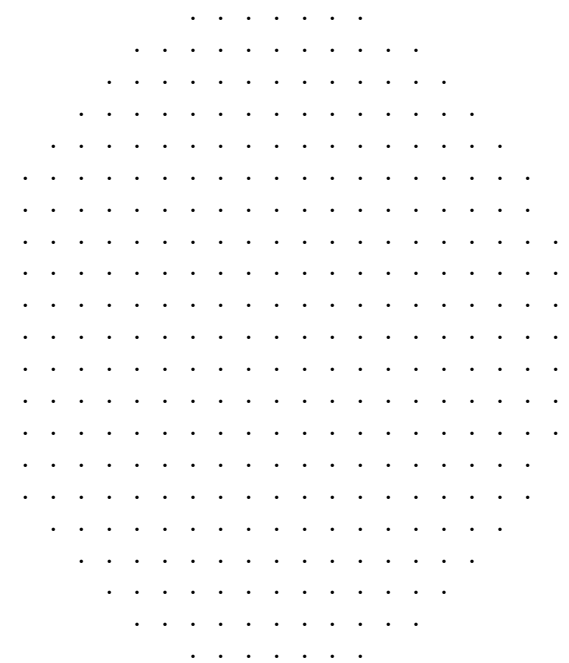
The circle function

Below is the VHDL function for the circle which returns a boolean when it is within the circle:

```
function f_is_circle_pixel(s_h ,s_v ,x ,y , size: integer)
   return boolean is
begin
   return (((x - s_h)*(x - s_h) + (y - s_v)*(y - s_v)) < (size*size));
end f_is_circle_pixel;
```

[1]The original project can be found on http://www.opencores.org/project,yavga
[2]Processor Local Bus, a bus interface from IBM

This is the output of the function in a C based testbench:



The rectangle function

Below is the VHDL function for the rectangle which returns a boolean when it is within the rectangle:

```
function f_is_rectangle_pixel(s_h,  s_v,  x,  y,  xwidth,  ywidth : std_logic_vector)
   return boolean is
begin
   return ((s_v >= y and s_v <= y+ywidth) and  (s_h >= x and s_h <= x+xwidth));
end f_is_rectangle_pixel;
```

Digit process

We have three of these placed in the top-right corner of the screen to display the score. The *digits* array contains the ROM describing the digits.

```
process (sys_clk)
   variable x_offset : std_logic_vector(10 downto 0);
   variable y_offset : std_logic_vector(9 downto 0);
   variable index : integer;
begin
   if (rising_edge(sys_clk)) then
      x_offset := hcount - x;
      y_offset := vcount - y;
      index := conv_integer((("0" & y_offset(9 downto 1)) AND NOT "0000000011") +
                            ("000" & x_offset(10 downto 3)));
      if ( hcount >= x AND hcount < (x + w) AND
```

```
            vcount >= y AND vcount < (y + h)) then
         if (value <= "1101") then
            enable <= digits(conv_integer(value))(31-index);
         else
            enable <= '0';
         end if;
      else
         enable <= '0';
      end if;
   end if;
end process;
```

3.2 Interrupt controller

The interrupt controller provides a convenient way of binding the interrupts to handler functions and is used by the mouse controller.

3.3 PS/2 Controller

The PS/2 Controller is the *XPS PS/2* controller found in the IP-catalog of the EDK[3]. There were no modifications done to the controller and using it was as simple as adding it in the *System Assembly View* and then connecting the ports.

3.4 Device Utilization

Below is the output from the hardware synthesis regarding device utilization.

```
Device utilization summary:
---------------------------
Selected Device : 3s1200efg320-4
 Number of Slices:                     1946  out of   8672    22%
 Number of Slice Flip Flops:           2030  out of  17344    11%
 Number of 4 input LUTs:               3183  out of  17344    18%
    Number used as logic:              2825
    Number used as Shift registers:     102
    Number used as RAMs:                256
 Number of IOs:                          26
 Number of bonded IOBs:                  26  out of    250    10%
 Number of BRAMs:                        17  out of     28    60%
 Number of MULT18X18SIOs:                 6  out of     28    21%
 Number of GCLKs:                         1  out of     24     4%
 Number of DCMs:                          1  out of      8    12%
```

4 Software

The software consists of a procedural game-engine and an interrupt-based mouse driver. The engine consists of an infinite loop that handles data from the mouse driver and the game logic.

[3]Embedded Development Kit. A software bundle from Xilinx

4.1 Game Logic

The game logic handles the ball and the racket and checks for any collisions between the ball and the walls and the ball and the racket. If any collision occur the direction of the ball is changed according to where the collision occured. A special algorithm is used for deciding the direction of the ball when the ball hits the racket, this ensures that the ball will bounce in different directions depending on where on the racket the ball bounces. A flowchart for the game logic can be seen in figure 3.

4.2 Mouse Driver

The mouse driver is interrupt-based and reads from the mouse controller once it signals that data is available. When four bytes have been read it sets a flag causing the main loop to process the new data. No data is processed in the mouse driver, the main loop is given the raw bytes sent from the mouse.

4.3 Hardware Communication

The software communicates with the hardware by writing to addresses on the PLB bus. Since the PLB interface is implemented by the Microblaze softcore it is very simple to write to the PLB addresses.

The EDK defines symbols that can be used instead of raw addresses which makes it as simple as doing `*XPAR_MY_IP_CORE_BASE_ADDR = value`. when writing to the PLB bus.

4.4 Memory Requirements

The memory requirements for running the actual game engine is pretty small, it only needs a few kilobytes. The size of the ELF[4] is 119502 bytes.

5 Problems and Solutions

1. The library path for the EDK kept duplicating itself and slowing down the project by loading the same library multiple times. This issue was solved by cleaning the library path manually in the Project Properties dialog in the EDK

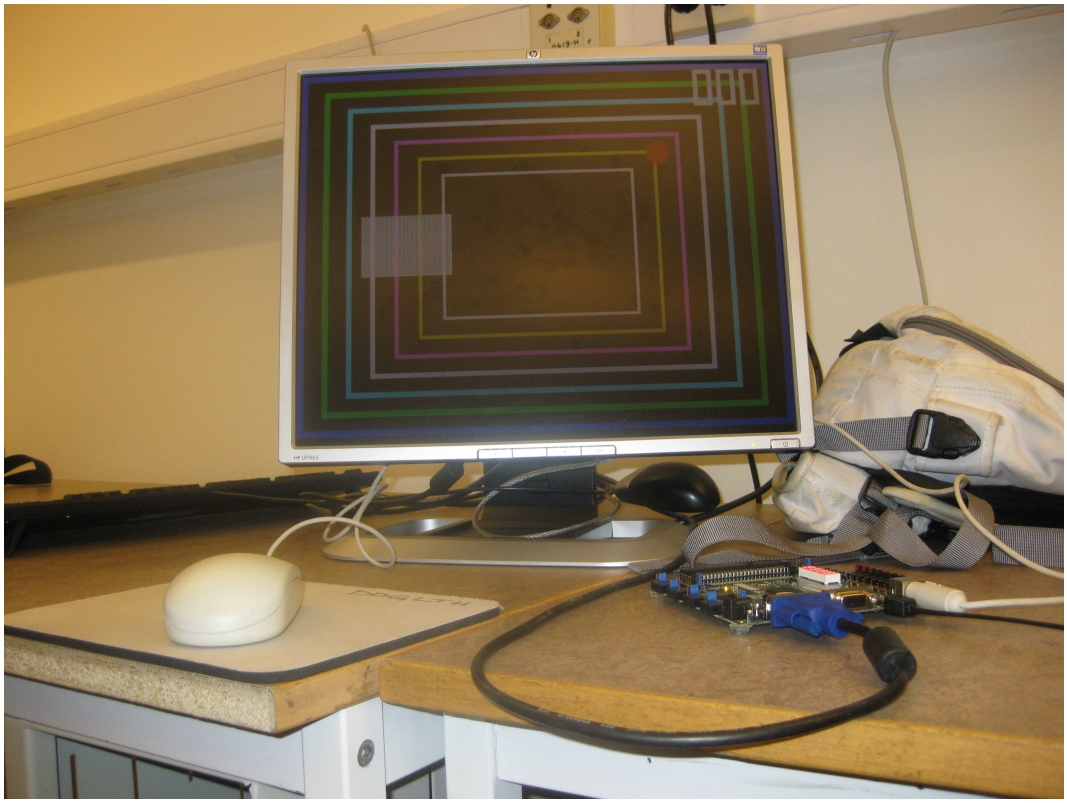
2. Custom PS/2 hardware caused issues with tri-state buffers on ports marked as both input and output. This issue resolved itself when we switched to the XPS PS/2 controller from Xilinx.

6 Conclusions

While doing this project we got some real experience with doing hardware/software co-design and integration. Overall it was a fun project and we think that the final result is satisfactory. We made a fun and challenging game and we learned a lot while doing it.

[4]Executable and Linkable Format, a file format for binary executables

7 Image



8 Credits

- Game engine by Daniel Raneland.
- VGA Controller from Open Cores and modified by Farooq Anwar and Christopher Abrahamsson with a few tweaks by Daniel Raneland.
- Mouse Driver by Christopher Abrahamsson and Farooq Anwar.

9 Instructions for use

To try Squashpongthingy out you need to download the project from http://users.student.lth.se/sx08fa9/Squa
It is then very simple to use the project, just unpack it to a directory of your choice and upload `download.bit` which is found in the implementation folder to your *Spartan 3E-1200* FPGA development board. Accessories needed to play are a screen connected to the VGA port and a mouse connected to the PS/2 port.

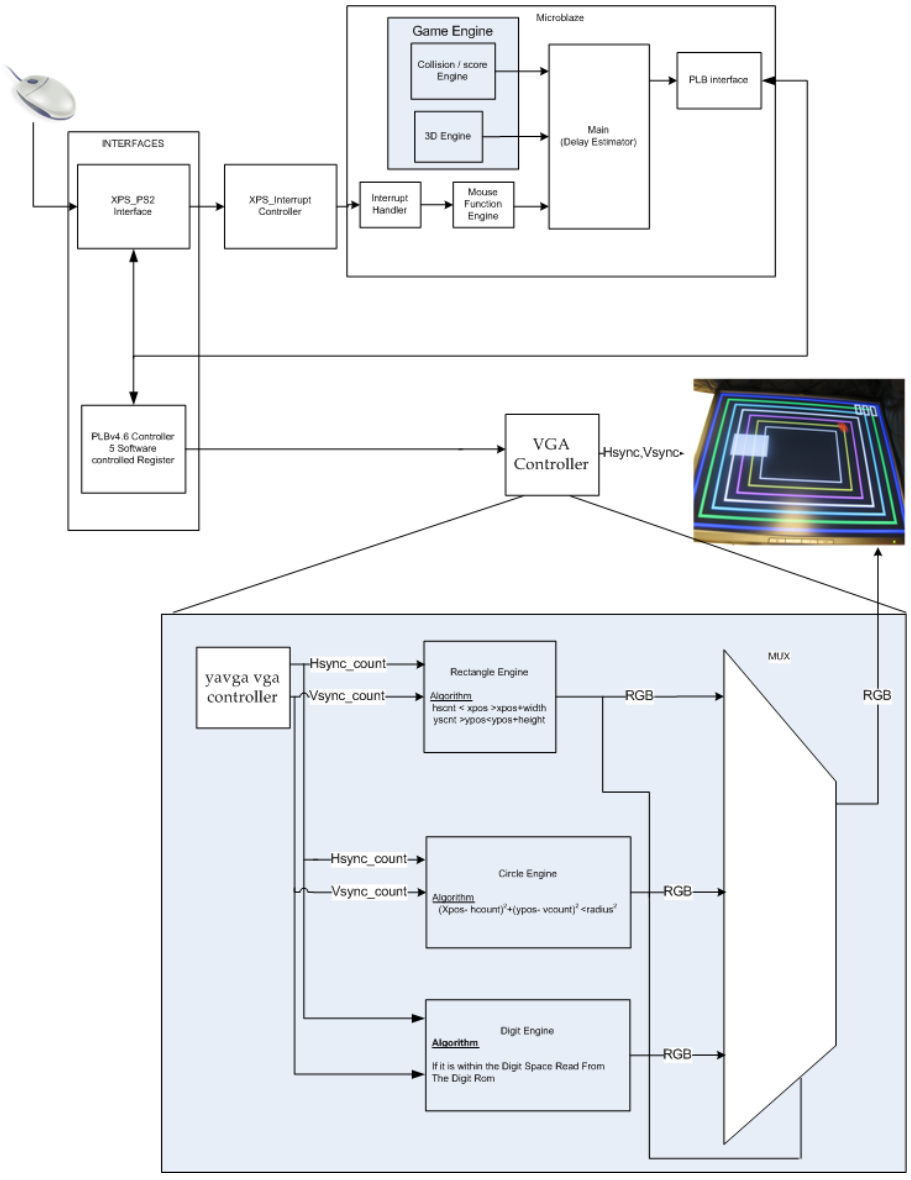


Figure 1: Overview of the system architecture

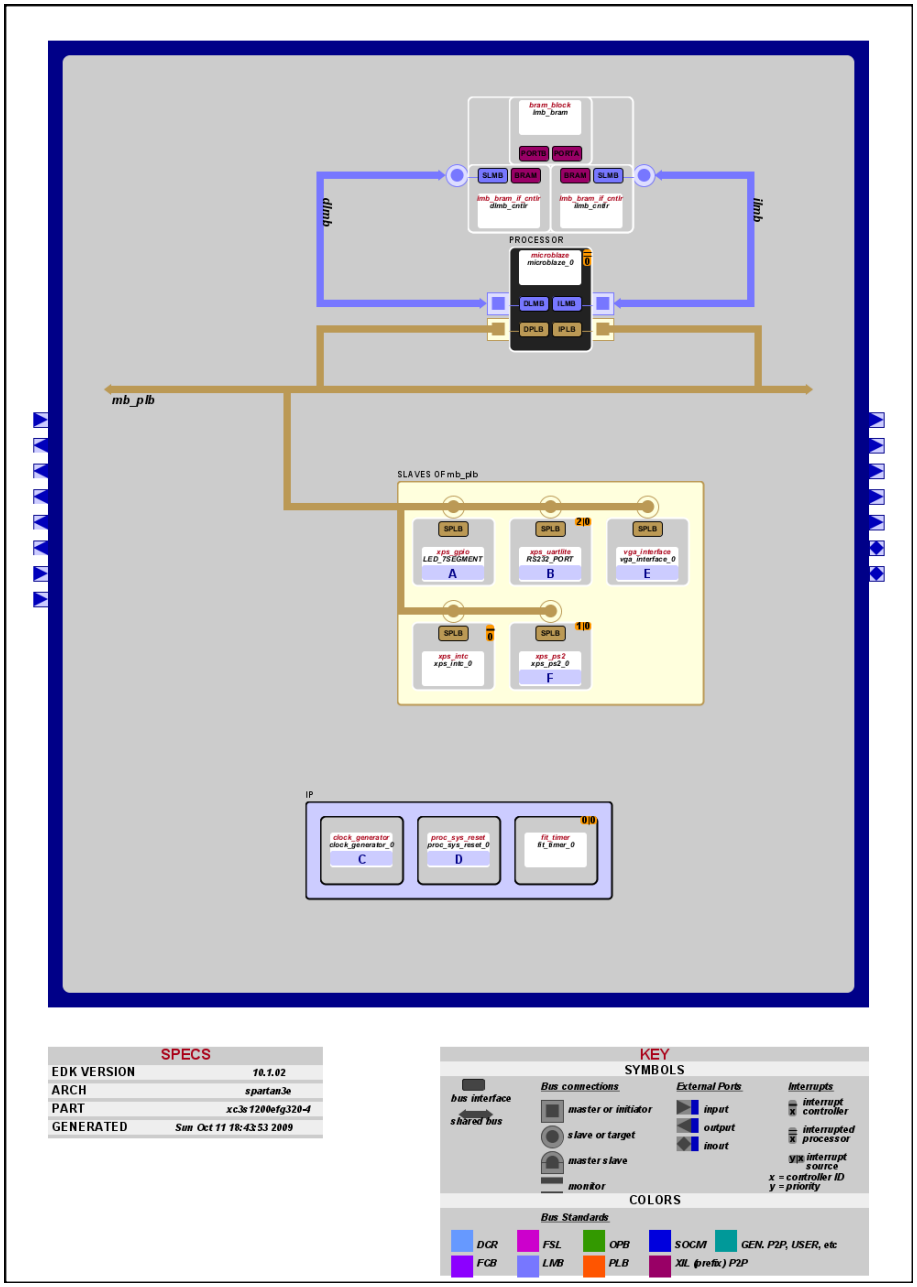


Figure 2: System Block Diagram

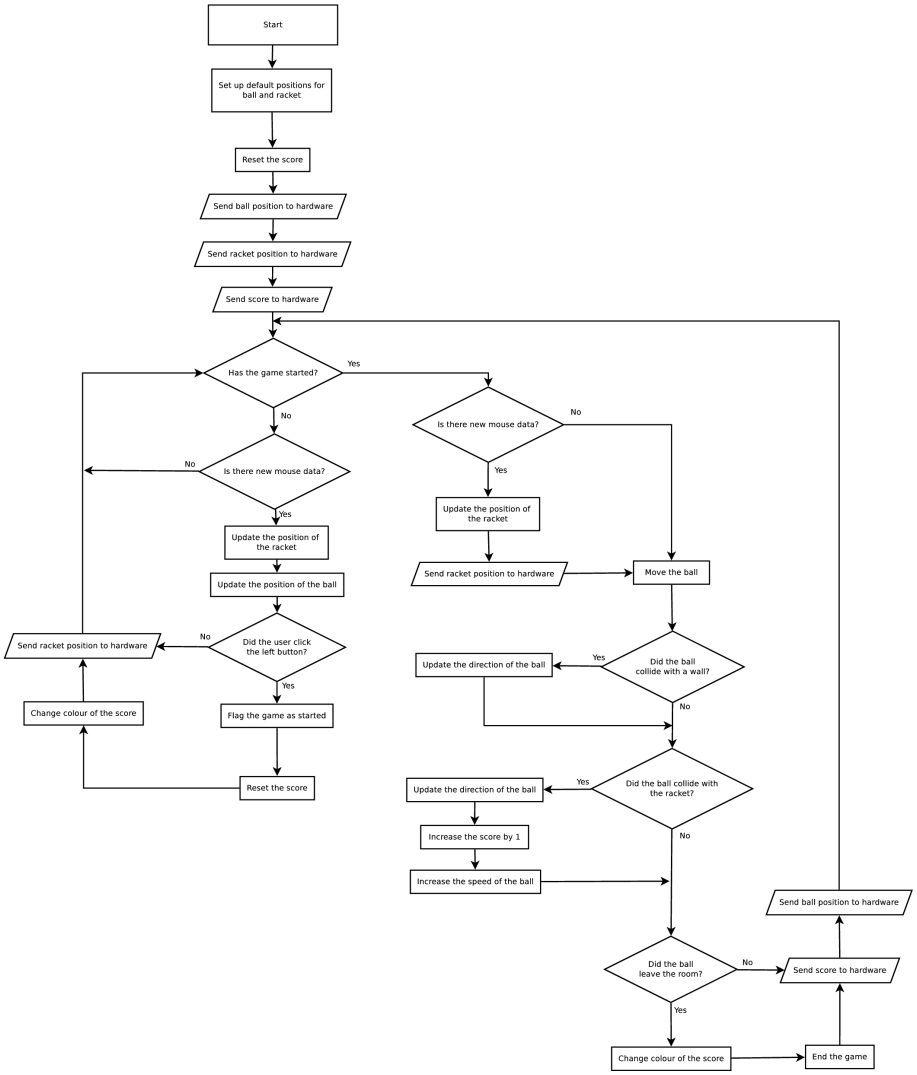


Figure 3: Flowchart for the game engine